УДК 32.81

О. И. Еськова (oieskova@rambler.ru),
канд. техн. наук, доцент

Р. Е. Гришечкин (kosmos2012@gmail.com),
студент

Белорусский торгово-экономический университет потребительской кооперации
г. Гомель, Республика Беларусь

МОДУЛЬНОЕ ТЕСТИРОВАНИЕ КАК ЧАСТЬ МЕТОДОЛОГИИ РАЗРАБОТКИ ПРОГРАММНЫХ ПРОДУКТОВ

В статье рассматриваются виды тестирования программного обеспечения и его роль в процессе разработки работоспособного кода. Выполняется обзор различных инструментов для модульного тестирования.

The article discusses the types of software testing and its role in the process of developing workable code. Are view of various unit testing tools is performed.

Ключевые слова: модульное тестирование; экстремальное программирование; рефакторинг; разработка через тестирование; фреймворки для модульного тестирования в Java.

Key words: unit testing; extreme programming; refactoring; test driven development; unit testing frameworks in Java.

Под тестированием кода понимается проверка соответствия между ожидаемым и реальным поведением программной системы. Тестирование может выполняться как самими программистами-разработчиками, так и специально обученными специалистами (тестировщиками).

В зависимости от анализируемых аспектов кода различают следующие виды тестирования:

– *Функциональное тестирование* – проверка того, что код адекватно выполняет свою задачу и соответствует спецификации на него. Проверяется правильность работы всех методов кода: возвращаемые значения, выбрасываемые исключения и изменения в состоянии системы после выполнения каждого метода.

– *Тестирование производительности* – оценка того, насколько быстро работает программа в обычных и стрессовых условиях (например, при большом количестве пользователей интернет-магазин не должен замедлять свою работу или становиться недоступным).

– *Тестирование удобства использования* обычно выполняется вручную специальным тестировщиком-пользователем. Такой тестировщик нажимает на кнопки, переходит по ссылкам, чтобы проверить работу интерфейса программы.

– *Тестирование безопасности* – проверка того, что код не дает потенциальной возможности доступа к несанкционированной информации, не позволяет испортить базу данных или препятствовать работе других пользователей.

Можно также представить следующую классификацию по уровням тестирования:

– *Модульное тестирование* (*юнит-тестирование*) – проверка работы отдельных модулей. Под модулем понимается один класс или группа тесно взаимосвязанных классов. Такой модуль рассматривается изолированно. Если же он зависит от других частей программы (например, обращается к базе данных или сетевому соединению), то на данном этапе зависимости закрываются специальными «заглушками». При этом считается, что окружение тестируемого модуля работает корректно. Этот вид тестирования обычно выполняется программистом-разработчиком класса. Обычно проверяется функциональность кода и иногда производительность.

– *Интеграционное тестирование* – проверка совместной работы нескольких модулей (необязательно всей системы в целом). Например, к протестированному модулю добавляется реально работающая база данных. Цель этого этапа – проверить информационные связи между модулями. Этот вид тестирования может выполняться программистами или тестировщиками в зависимости от политики руководства компании-разработчика.

– *Системное тестирование* – это работа системы в целом. Система должна быть помещена в то окружение, где она будет эксплуатироваться, все компоненты должны быть реальными и уже прошедшими модульное тестирование. Обычно выполняется тестировщиками.

Выделяют следующие подходы к тестированию:

– Тестирование «черного ящика». Тестировщик не знает, как устроен код, а создает набор тестов только на основе спецификации к программе.

– Тестирование «белого ящика». Тестировщик знает, как код устроен, при разработке тестов может проверять, в том числе, внутреннее состояние системы, приватные методы. В качестве тестировщика в этом случае выступает программист-разработчик кода.

Модульное тестирование (юнит-тестирование) должно сопровождать процесс разработки любого программного продукта. Разработка тестов (тесты – это специальный код (классы и методы) требует определенных усилий и затрат времени.

Использование тестов поощряет программистов вносить изменения в код: добавлять новую функциональность или проводить рефакторинг. Если после внесения изменений предыдущие тесты выполняются успешно, то это служит доказательством того, что код работоспособен. Это позволяет фиксировать работоспособные варианты продукта в системе контроля версий.

Упрощается интеграция отдельных модулей. Если есть уверенность, что каждый модуль по отдельности работоспособен, а при интеграции возникают ошибки, следовательно, проблема заключается в связях между ними, которые нужно проверить.

Тесты представляют собой особый вид документирования кода. Если программист-клиент не знает, как использовать данный класс, он может обратиться к тестам и увидеть там примеры использования методов этого класса.

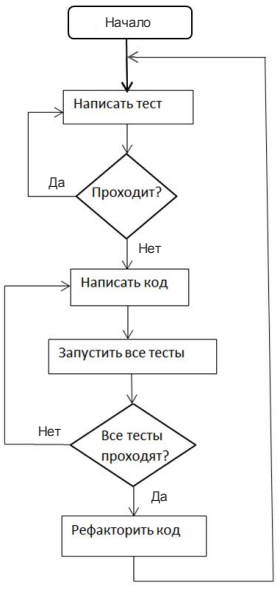
Использование модульного тестирования повышает качество разрабатываемого кода. На большой и запутанный код трудно написать тест. Это заставляет программиста инкапсулировать отдельные фрагменты кода и отделять интерфейс от реализации. Методы тестируемого класса становятся проще.

С тестированием связан ряд приемов методологии экстремального программирования. Экстремальным программированием называется совокупность приемов организации работы разработчиков кода, которые позволяют сократить сроки разработки программного продукта, уменьшить стресс и сделать процесс разработки более предсказуемым. Наиболее популярные методики экстремального программирования – это регрессионное тестирование и TDD (разработка через тестирование).

Регрессионное тестирование – это собирательное название для всех видов тестирования, направленных на обнаружение ошибок в протестированных участках кода. Если после внесения изменений в программу перестает работать то, что должно было продолжать работать, то возникает регрессионная ошибка (regression bug). Такие ошибки находятся, если после каждого изменения модуля проходит весь набор тестов, ранее созданных для этого модуля.

Разработка через тестирование (TDD – test-driven development) – одна из практик экстремального программирования, которая предполагает разработку тестов до реализации кода [1]. Сначала разрабатывается тест, который должен быть пройден, а потом – самый простой код, который позволит пройти этот тест. Алгоритм действий при реализации TDD представлен на нижеприведенном рисунке.

Алгоритм методики разработки через тестирование



Один цикл разработки методом TDD состоит в следующем:

– Из репозитория извлекается модуль, на котором уже успешно выполняется некоторый набор тестов.

– Добавляется новый тест, который не должен проходить (он может иллюстрировать какую-то новую функциональность или ошибку, о которой стало известно). Этот шаг необходим также для проверки самого теста.

– Изменяется программа так, чтобы все тесты выполнились успешно. Нужно использовать самое простое решение, которое не изменяет предыдущие тесты.

– Выполняется рефакторинг кода, после которого тесты тоже должны работать. Рефакторингом называется улучшение структуры кода без изменения его внешнего поведения. Например, переименовываются методы для лучшей читабельности программы, устраняется избыточный, дублирующий код, инкапсулируется поле, выделяется отдельный класс или интерфейс и т. п.

– Весь комплект изменений вместе с тестами заносится в репозиторий (выполняется операция commit).

Таким образом, модуль всегда поддерживается в стабильном работоспособном состоянии.

Для каждого языка программирования существуют свои инструменты для создания и использования модульных тестов (юнит-тестов). Для языка Java эти инструменты можно условно разделить на три группы:

– Фреймворки для написания и запуска тестов: JUnit, TestNG.

– Библиотеки проверок: FEST Assert, Hamcrest, XMLUnit, HTTPUnit.

– Библиотеки для создания тестовых дублеров – Mockito, JMock, EasyMock.

Библиотеки проверок позволяют расширить возможности проверок результатов тестов (которые имеются в базовом фреймворке, например, в JUnit), а также сделать результаты логирования тестов более удобными.

Библиотеки для создания тестовых дублеров позволяют упростить написание «заглушек» для внешних по отношению к тестируемому модулей. Такие «заглушки» носят название моки (mock-object) и стабы (stub-object). Stub – более примитивный объект, просто заглушка. В лучшем случае может печатать трассировочное сообщение. Mock более интеллектуален и может реализовать какую-то примитивную логику имитации внешнего объекта.

Аналогичные JUnit библиотеки классов существуют и в других языках программирования. Все они имеют общего «предка» – библиотеку классов SUnit для языка SmallTalk. Эта библиотека потом была скопирована в Java под названием JUnit, а оттуда в C++ с именем CppUnit и в Ruby как RUnit (потом она была переименована в RSpec). Также из Java эта библиотека переместилась в Python под названием unittest. Аналогичные инструменты для языка C# носят название XUnit.net и NUnit [2].

Таким образом, многие инструменты тестирования в разных языках программирования имеют одно происхождение и общее имя xUnit. В книге «xUnitTestPatterns» приводится множество примеров того, как работать с фреймворками этого семейства [3].

Модульное тестирование – часть культуры программирования, позволяющей разработчику выглядеть профессионалом своего дела. Навыки юнит-тестирования требуются в большинстве публикуемых вакансий и придадут вес резюме в любой сфере разработки программного обеспечения.

Список использованной литературы

1. **Бек, К.** Экстремальное программирование. Разработка через тестирование / К. Бек. – СПб. : Питер, 2017.

2. **Ошероув, Р.** Искусство автономного тестирования с примерами на C# / Р. Ошероув. – М. : ДМК Пресс, 2014.

3. **Meszaros, G.** xUnit Test Patterns. Refactoring Test Code / G. Meszaros // USA : Addison-Wesley, 2007.